

# Mirage: Towards Deployable DDoS Defense for Web Applications

Prateek Mittal  
University of California, Berkeley  
pmittal@eecs.berkeley.edu

Dongho Kim  
Cisco Systems  
dkim99@illinois.edu

Yih-Chun Hu, Matthew Caesar  
University of Illinois at Urbana-Champaign  
{yihchun,caesar}@illinois.edu

**Abstract**—Distributed Denial of Service (DDoS) attacks form a serious threat to the security of Internet services. However, despite over a decade of research, and the existence of several proposals to address this problem, there has been little progress to date on actual adoption. We present *Mirage*, a protocol that achieves comparable performance to other DDoS mitigation schemes while providing benefits when deployed only in the server’s local network and its upstream ISP, where local business objectives may incentivize deployment. *Mirage* does not require source end hosts to install any software to access *Mirage* protected websites.

Our approach is that end hosts can thwart the attackers by employing the principle of a moving target: end hosts in our architecture periodically change IP addresses to keep the attackers guessing. Knowledge of an active IP address of the destination end host can act as an implicit authorization to send data. We evaluate *Mirage* using theoretical analysis, simulations and a prototype implementation on PlanetLab. We find that our design provides a first step towards a deployable, yet effective DDoS defense.

## I. INTRODUCTION

Denial of service (DoS) attacks form a serious threat to the security of Internet services. In a DoS attack, a single computer or a group of computers (DDoS) flood the victim’s machine by sending a large number of packets, exhausting its bandwidth capacity (or software processing capabilities). This causes legitimate traffic to back off, resulting in denial of service to properly behaving hosts. The frailty of today’s Internet to DDoS attacks is evident by recent attacks on entire countries such as Georgia and Estonia, whistle-blowing websites such as Wikileaks, financial transaction hubs such as Mastercard, Visa, and PayPal, intelligence agencies such as CIA, government organizations such as US department of justice, and online election servers [18], [47], [51].

There are two main schools of thought regarding DDoS defense. *Filtering*-based approaches [10], [32], [34] first aim to detect and classify the malicious traffic and then install filters at the network layer to prevent the malicious traffic from exhausting victim’s resources. On the other hand, *Capability*-based approaches [38], [52], [53] aim to send only authorized traffic to hosts in the Internet. Extensive academic work has been done on these approaches, and their tradeoffs are well understood [32]. However, despite over a decade of research, little progress has been made on real-world deployment. In fact, the number of DDoS attacks increased by 57% in 2011 [30]. A fundamental obstacle to deployment is that a deploying AS does not benefit until other ASes also

participate in the scheme. This dependency between ASes is a chicken-and-egg problem for deployment in the Internet, which traditionally does not force a protocol to be deployed. Phalanx [17] aims to reduce the number of ASes that are required to deploy the protocol for effective defense, but does not completely solve the chicken-and-egg problem (see § II).

In this paper, we propose *Mirage*, a DDoS defense mechanism that aims to remove the required deployment at other ASes as much as possible. Towards this end, we adopt an approach similar to *frequency hopping* in wireless networks [5]: in *Mirage*, a destination end host varies its IP addresses according to a pseudorandom sequence known only to authorized hosts. Knowledge of an *active* IP address owned by the destination end host acts as an implicit authorization to send data, and enables *Mirage* to reduce the required deployment of other ASes. To realize this idea in a concrete system, we leverage known ideas: computational puzzle [21], [49], [50], filtering and fair queueing. Our key contribution is a system architecture that integrates these existing primitives with the novel paradigm of IP address hopping, with the goal of reducing the need for deployment across organizational boundaries. The security of *Mirage* depends on the space of IP addresses that a destination end host can use. To get a sufficiently large space of IP addresses, each *Mirage*-protected server uses its IPv6 prefix and chooses interface addresses as these secret addresses. The practicality of *Mirage* will grow each day with the increasing deployment of IPv6 and the ability of IPv4-only-connected hosts to use IPv6 through tunneling, as described in § III-A. Randomly hopping IP addresses play the same role as capabilities in capability-based schemes, which in contrast to *Mirage*, introduce a completely new packet header [38], [52], [53]. It is much more likely that IPv6 will witness widespread deployment, as opposed to a new custom Internet header.

We evaluate *Mirage* using theoretical analysis, large-scale simulations and a prototype implementation on PlanetLab. We find that *Mirage* is able to provide honest nodes with their fair share of system resources, comparable to previous DDoS mitigation mechanisms. At the same time, *Mirage* is able to provide some benefits to a server when only the server’s local network and its upstream ISP deploy it. Unlike previous proposals, *Mirage* only requires functionality from routers that is already deployed in today’s routers, though this functionality may need to be scaled depending on the point of deployment. In particular, *Mirage* does not

require router hardware or software modifications to support new headers/fields in network packets, and achieves this without changing the semantics of existing header fields. Our architecture also does not require any cryptography at the routers. Finally, source end hosts can take advantage of Mirage-protected sites without installing any software. We find that our design provides a first step towards a deployable, yet effective DDoS defense.

## II. RELATED WORK

Our work is related to two main schools of thought about DDoS mitigation: *filtering* and *capability*-based designs.

**Filtering and rate limiting:** Pushback [34] defends against DDoS attacks by having congested routers rate limit the set of flows responsible for congestion (aggregate rate limiting). If local rate limiting is not sufficient to mitigate congestion/DDoS attacks, then the router contacts its upstream router and asks it to perform rate limiting as well. AITF [10] is a traffic filtering architecture that leverages record routing techniques to enable a victim to identify routers close to a source. The victim can then install filters close to the source of attack traffic. StopIt [32] proposes a filtering mechanism whereby a receiver can request a particular flow to be blocked for a period of time. While filtering-based schemes are powerful, they face significant practical hurdles, as they assume the ability to differentiate attack traffic from legitimate traffic. Adversaries may make use of cover traffic to hide their communication, or may mimic legitimate traffic (e.g., by mimicking legitimate GET requests). Mirage does not make any assumptions about the ability to classify malicious traffic; however, if a method to differentiate attack traffic is available, Mirage can leverage it to provide a significant subset of the requisite filtering capability.

**Capabilities:** Capability-based schemes provide the receiver an ability to directly control its reachability within the network. When a sender wants to transmit packets to a receiver, the receiver decides whether to permit it and lets routers know its decision. Then, the router allows the corresponding packets to pass. SIFF [52] gives a technique to efficiently allow receivers to authorize senders to transmit. A router sends tokens to a receiver that can be used to authorize sender requests. When a receiver wants to receive packets from the sender, the receiver passes the token to a sender. The sender adds the token to packet headers it transmits to get a preferential service from the router. TVA [53] refines SIFF to defend against brute-force attacks to improve practicality and efficiency. Portcullis [38] addresses a vulnerability in both SIFF and TVA where an attacker could launch a denial of capability attack, and prevent the initial request packets of honest users from reaching the victim. Portcullis makes use of computational puzzles to force requesting users to perform work before being able to access the initial request channel. In § VIII, we discuss the deployment challenges for capability-based schemes, including the problem of low

benefit for early adopters, the need to upgrade client end-host software, requiring new router primitives such as cryptographic support, as well as requiring a new custom Internet header. Additionally, we note that in most of the above approaches, the capability to send a packet is specific to the path used to route a packet to a destination. Thus, route changes in the Internet pose a significant challenge for these approaches. In contrast, Mirage uses the knowledge of an active destination IP address within the IP address range of the destination end host as an implicit authorization to send data. Such an authorization mechanism is end-to-end, and is not affected by route changes in the Internet.

**Other approaches:** Phalanx [17] proposes to use a swarm, a large pool of geographically and topologically distributed well provisioned machines to absorb the DDoS attack, and forward legitimate traffic to the victim. Once a few ASes deploy an adequate number of such well provisioned machines to form an overlay network, any ISP can gain immediate benefit from deploying Phalanx. While the design of Phalanx is a significant improvement over prior work in this domain, it still requires other ASes to deploy the protocol and participate in an overlay network. Wang et al. [49], [50] have also proposed the use of client puzzles, but their approach does not focus on deployment. NetFence [33] proposes to use the network as the first line of defense to mitigate DDoS attacks, but assumes that IP spoofing is not possible. Speak-up [48] uses proof of work techniques for DDoS defense, but its focus is on application-layer attacks. Similarly, the end-to-end approach of Gligor [20] is also limited to application-layer attacks. Kandula et al. [29], and Morein et al. [35] propose defense mechanisms wherein users solve CAPTCHAs [6] to differentiate themselves from bots. Mirage does not require users to solve CAPTCHAs. Keromytis et al. [31] propose an approach where only select nodes in an overlay network are allowed to communicate with a destination, and their IP addresses are kept hidden. Pre-authorized users are given knowledge of a subset of those select nodes, and can use them as a proxy to communicate with a destination. Mirage builds upon the idea of hiding proxy IP addresses by introducing the notion of destination IP address hopping. Mirage does not require the use of proxies or overlay networks, and provides service to all users (not just pre-authorized users). Similar to Mirage, Stavrou et al. [43] also propose the use of spread-spectrum techniques, but they rely on an external overlay network, and do not focus on the issue of deployment.

**Moving target defenses:** Mirage is inspired from frequency hopping in wireless networks, and more generally, it is related to the concept of moving target defenses [28] which aim to create uncertainty for the adversary. Mirage's notion of IP address hopping is related to the concept of address space randomization techniques such as ALSR [41], which aims to defend against untrusted code, and NASR [9], which aims to defend against hit-list worms, and the work of Shue

et al. [42], which aims to implement network capabilities. The key challenges in the application of address space randomization techniques for DDoS defense are to (a) enable (any) legitimate client to access the server, (b) constrain the effects of malicious clients, and (c) defend against attacks on upstream network prefixes. Mirage solves these challenges, with the additional goal of reducing the need for deployment across organizational boundaries. In contrast, both NASR and the work of Shue et al. are vulnerable to malicious clients that query DNS servers to learn victim server’s address; unlike Mirage, they lack proof-of-work mechanisms such as computational puzzles to limit attackers. Moreover, both honest and malicious clients are issued identical addresses for the victim server, enabling malicious clients to dominate victim’s network resources. Finally, both NASR and Shue et al. do not defend against DDoS attacks on victim’s upstream network prefixes, for example, an adversary can flood an IP address that was previously in use by the victim.

### III. MIRAGE OVERVIEW

We next describe some goals and limitations of our work (§ III-A), and explain how we design Mirage to achieve these goals (§ III-B).

#### A. Design Goals and Limitations

1. *Incremental and incentivized deployment:* We target an incrementally deployable design where a small number of routers should be able to deploy Mirage and bring immediate benefit to downstream servers. We note that in such a design, local business objectives may incentivize deployment. Our architecture should not interfere with operation of existing or non-upgraded network protocols or systems. Mechanisms that introduce additional packet headers or protocol layers may not satisfy this property. For instance, schemes that introduce additional headers can face significant challenges during incremental deployment, due to incompatibilities with scrubbing and IDS services, layer-7 and layer-4 load balancing, and other middleboxes and network services that inspect non-IP layers of the network stack.

2. *Lowering deployment cost:* We target a design which minimizes the requirement for cooperation across administrative or trust boundaries, and does not rely on an external overlay network to deploy the mechanism. Mechanisms that require the use of trusted hardware [8], or require end users to upgrade software face major deployment hurdles and are incompatible with our design goals. Similarly, we avoid reliance on new primitives from routers, such as router cryptography and additional packet headers, which require significant support from vendors and increase the cost of network equipment.

3. *Network fairness:* In the absence of techniques to classify attack traffic, we aim to provide each user with its fair share of the network. However, if techniques to partially classify attack traffic are available, then Mirage can leverage them.

4. *Low overhead:* We target a design that does not impose additional cost on the network or end hosts when the system is not under attack (adaptive defense).

**Limitations:** Our work has several limitations. First, like other schemes that use puzzles, such as Portcullis [38] and Phalanx [17], Mirage requires the ability to distribute cryptographic information (puzzles) in a manner that is not subject to DDoS. Mirage can use existing replicated services such as Akamai or well provisioned cloud services such as Amazon S3 or even DNS. Second, Mirage requires a large IP address space to perform effectively. To achieve this, Mirage can make use of IPv6. The Internet is already in a transition phase to IPv6 [3], and we note that IANA has already exhausted its pool of IPv4 allocation blocks in February 2011 [2]. Mirage is able to accommodate scenarios where source end hosts are in an IPv4 only network, by the use IPv4-IPv6 translators. For example, Teredo IPv6 tunneling [44] is already built into Windows. Mirage can even protect destination end-hosts serviced by an IPv4 network; in such scenarios, the victim sets up a tunnel to a IPv6 provider that supports Mirage. To protect against native IPv4 DDoS, the victim either keeps its IPv4 address secret, or requests its ISP to block any traffic that does not originate from the tunnel server. We emphasize that in contrast to Mirage’s approach of leveraging IPv6, prior work on capability advocates a completely new packet header.

Third, our design requires loose time synchronization, on the order of tens of seconds. To address this, Mirage may require external systems such as NTP, which can provide time synchronization with accuracies on the order of hundreds of milliseconds on wide area networks [24]. Fourth, Mirage targets per computation fairness. If the ratio of honest sources’ computational power to the total computational power of all nodes (honest nodes and attacker nodes) is  $r$ , then Mirage provides the honest nodes with a fraction  $r$  of the bottleneck link bandwidth. In scenarios where there is a mismatch between the computational resources of the attacker and the honest nodes (for example, a webserver with a small user base being attacked by a very large botnet), the fair share of the honest nodes will be small.<sup>1</sup> We note that this limitation is also shared by other state of the art mechanisms that rely on computational puzzles, such as Portcullis [38] and Phalanx [17]. Another aspect related to the issue of resource mismatch is the differences in computational capabilities of various devices, such as a smartphone and a GPU. However, Mirage is compatible with the use of memory-bound puzzles [4], which have been shown to lower resource disparity between devices.

**Threat model:** There are two types of DoS attacks: (a) network-layer attacks, where an attacker attempts to overwhelm the transmission capabilities of the underlying

<sup>1</sup>Even though the fair share of honest nodes may be small in some scenarios, their requests will still succeed.



network, and (b) application-layer attacks, where the attacker attempts to overwhelm processing capabilities of the victim’s application software. While the focus of this paper is on network-layer DoS attacks, Mirage provides an interface to the application layer to fair queue incoming requests based on proof of work (destination IP addresses). Similar to Portcullis [38] and Phalanx [17], we assume that lookup services such as DNS can be highly replicated (since they serve short, static content), and are not subject to DDoS. We assume the adversary may have access to a large number of hosts (e.g., via ownership of a botnet), and may perform IP spoofing. We assume routers can be compromised, though we note that a compromised router can always block traffic towards the victim.

### B. System architecture

*IP address hopping:* To mitigate denial of service attacks, we need some way to make it harder for the attacker to reach the server. In previous approaches, capabilities or randomly chosen proxies (Phalanx [17]) are used for that purpose. However, these approaches add a new packet header and require deployment in other ASes thereby hindering incentivized deployment. Mirage’s approach is to use randomly changing destination IP addresses. This idea enables Mirage to avoid the necessity of other ASes’ deployment. Mirage consists of a set of add-ons to existing Internet services (Figure 1) that enable a server to dynamically *hop* (change) its IP address. In particular, the server is assigned a *set* of IP addresses, and repeatedly modulates the address it uses from this set via a deterministic pseudorandom function. This function computes the server’s current IP address given the current time as input. The server can then share that pseudorandom function with authenticated clients; these clients can then determine the IP address used by the server by computing the pseudorandom function.

*Slowing the attacker with computation-limited hopping functions:* While certain Internet services can authenticate clients (e.g., systems that use CAPTCHAS or require the user to sign in with a secret password), other services may be unable to distinguish valid user requests from malicious requests. To support these services, a form of fair sharing mechanism is necessary. Options for fair sharing include bandwidth-based fairness [48] or computation-based fairness [38]. We do not use bandwidth-based fairness since it may induce congestion collapse in network. Mirage makes it more computationally difficult for the client to retrieve the IP address using the pseudorandom function. This is done by having the server construct a computationally-harder version of the pseudorandom function, which is then handed out to clients. This computationally-harder version is constructed by incorporating a cryptographic puzzle into the pseudorandom function returned to the client. Mirage makes use of hard-to-DoS infrastructures such as Akamai, DNS or well provisioned cloud services such as Amazon S3 for puzzle distribution. It

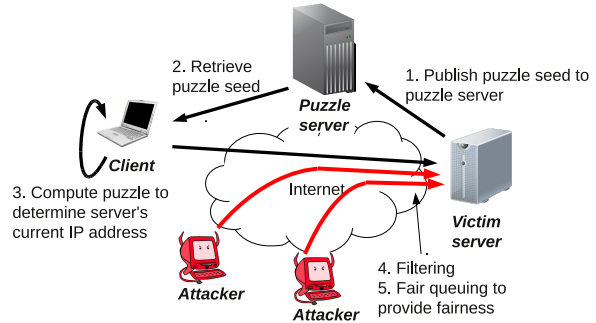


Fig. 1. Mirage network architecture

is important to note that putting all servers to cloud services to defend against DDoS attacks is expensive. Mirage uses such service for only puzzle distribution.

*Avoiding network bottlenecks with in-network filtering:* Even though a server can perform IP address hopping, an adversary can still attack the server by sending traffic to any of the alternate IP addresses assigned to the server, even though they are not in use. To address this, Mirage requires routers to perform in-network filtering. In particular, the victim host may instruct upstream routers of IP address ranges that it is currently not using. This may be done by a variety of mechanisms. For simplicity, our design assumes that the end host explicitly publishes access control lists that are serviced by upstream routers. Ideally, this filtering should be done near bottleneck links.

*Isolating the attacker with address sets:* Hopping IP addresses to evade the attacker only works when the attacker does not know the new IP address of the server. However, preventing the attacker from knowing an active IP address of the victim seems difficult, while still enabling non-malicious clients to know the current IP address. Hence, Mirage instead attempts to constrain the effects of malicious clients by associating a *set* of active IP addresses with the server, and and returning different elements of that set to different clients. To achieve this, we leverage standard techniques (such as those used by DNS [7]) to return different IP addresses to different clients based on their topological location. To defend against attacks where a single malicious client solves a computational puzzle and shares the puzzle solution with its colluding attackers, we need to provide isolation across elements of the set. To do this, the server or the bottleneck link can utilize fair queuing.

An example of the Mirage protocol is shown in Figure 1: (1) First, the server determines a seed for a globally-known pseudorandom function, and registers it with its puzzle server. (2) The server then begins using that pseudorandom function to compute its currently active set of IP addresses. This function is periodically recomputed to perform the hopping. (3) When a host wishes to make a request, it does a DNS lookup for the victim, gets redirected to the puzzle server, and retrieves the computational puzzle. The user then executes the puzzle to determine the server’s current IP address. The user

then sends packets to the server. The user periodically re-executes the puzzle to keep track of the server’s current IP address. (4) When a malicious host, such as a bot, wishes to execute a request, it follows exactly the same procedure. Due to the puzzles, the botnet cannot acquire more IP addresses than its computational power. Finally, the network filters traffic that does not have a victim server’s current IP address.

Due to the per-destination fair queuing, bandwidth is shared fairly across the destination IP addresses, reducing power of an attacker.

#### IV. PROTOCOL DESCRIPTION

In this section, we describe the basic Mirage design. In particular, we describe the higher level functions performed by destination end hosts, network routers, puzzle servers, and source end hosts. We discuss security/performance improvements, as well as deployment details of our protocol in § V.

##### A. Destination end hosts

**IP address hopping:** Our core mechanism is inspired by *frequency hopping*, used in wireless networks. Frequency hopping is a way to transmit radio signals by rapidly switching the carrier through a pseudorandom sequence of channels. If a receiver knows the pseudorandom sequence (e.g., if it shares a key with the sender), it can listen on the same sequence of frequencies to receive the transmission. Doing this can make wireless devices less susceptible to “jamming” attacks, which attempt to deny service by transmitting undesired traffic on a communication channel.

In our work, we apply this idea to make end hosts less susceptible to denial of service attacks. In particular, end hosts are assigned a consecutive range (prefix) of IP addresses. The sender and receiver then periodically hop through a pseudorandom sequence of IP addresses when communicating. We assume IPv6, to improve feasibility of allocating multiple IP addresses to end hosts, given IPv6’s larger address space (IPv6 addresses are 128 bits, with 64 bit subnet/interface addresses).<sup>2</sup> More generally, instead of having a single active IP address, end hosts can choose to have a *set* of active IP addresses from amongst their allocated range. This set of active IP addresses will periodically change. Each end host performs IP address hopping using a local cryptographic master key. At any instance of time, the master key can be used to determine the set of active IP addresses for that end host:

$$IP(i) = PREFIX || H(ENC_{KEY}(i || TIME))$$

where  $IP(i)$  denotes the end host’s  $i$ ’th active IP address,  $PREFIX$  is the prefix IP address associated with the end host,  $||$  represents the concatenation operation,  $H$  is a

<sup>2</sup>Mirage permits the same number of IPv6 devices as in the current IPv6 deployment model, where devices are assigned a unique 64-bit interface address (RFCs 3513 and 4291 [22], [23]).

cryptographically secure hash function with output length  $128 - |PREFIX|$  bits, and  $ENC_{KEY}(x)$  is the encryption of  $x$  with the key  $KEY$ . This set of active IP addresses is kept a secret, and will only be used under a DDoS attack. In addition to this secret set of IP addresses (which keep changing), the end hosts also maintain a static (not hopped) IP address, which is used when the host is not under attack.

**Puzzle server redirection:** As in the current Internet architecture, destination end hosts set up a DNS entry for their hostname. Under a DDoS attack, a destination end host (victim server) re-registers with its authoritative DNS (ADNS) to point its record to the puzzle server, using similar techniques to those used in CDNs for redirection and load balancing (e.g., small TTL). From that point onwards, the source end hosts are redirected to the puzzle servers.

**Filtering requests to routers:** Under a DDoS attack, the victim communicates the current set of active IP addresses to its upstream ASes. The upstream ASes can then block incoming traffic to any of the remaining IP addresses which are not in the active set. Thus, knowledge of any active IP address can be used as an implicit authorization to send data to that destination end host.

**Leveraging attack traffic identification:** Note that in the absence of the ability to identify attack traffic, Mirage aims to provide per computational fairness to clients. However, if it is possible to identify attack traffic, then Mirage can leverage this ability to improve performance for honest clients. In particular, the victim server can stop using the destination IP addresses corresponding to attack traffic, and request the upstream ASes to filter those IP addresses.

##### B. Routers

**Filtering traffic with existing router interfaces:** Whenever a victim is under a DDoS attack, the upstream routers receive information about the set of active IP addresses currently in use by the victim. The upstream routers can then filter out incoming traffic to all remaining (non-active) IP addresses (as well as other IP addresses that are identified by the victim as being associated with attack traffic). Note that the victim can start by installing filters at its edge router, and continue this process at the routers further upstream until its link is no longer congested. In § V we show that the upstream ASes can perform such filtering using *existing* router interfaces such as access control lists (ACLs) configured via iBGP feeds from IRSCP [46] style management servers. Moreover, in § VII, we show that performing such filtering only at the server’s upstream provider is sufficient to block most attacks.

**Per-destination fair queueing:** After filtering out packets with non-active destination IP addresses, the upstream routers perform fair queueing per destination. This results in per destination address fair allocation of resources when attack traffic cannot be distinguished from legitimate traffic. We note that routers already support fair queueing as an option,

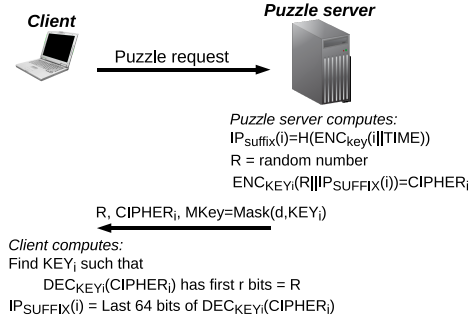


Fig. 2. Puzzle construction: active IP address of the server is the solution.

easily enabled via configuration changes. We discuss this in more detail in § V.

### C. Puzzle Servers

Under attack, source end hosts are redirected to a puzzle server.

**Active IP address generation:** The puzzle server shares a cryptographic key with the destination end host. The puzzle server can use the end host’s cryptographic key to derive a set of the end host’s active IP addresses at any instance of time (as discussed in § IV-A).

**Computational puzzles:** When a source end host first contacts the puzzle server, it responds with a computational puzzle. Upon receiving a solution to the computational puzzle from the requesting node, the server sends back an ephemeral active IP address of the destination end host. We use Portcullis [38] to implement the computational puzzles aspect.

### D. Source end hosts

The source end hosts first contact the DNS server to resolve the hostname for the destination end host. If there is no DDoS attack on the destination, they receive the conventional static non-hopping IP address for the destination. On the other hand, the source end hosts are redirected to a puzzle server during a DDoS attack. The source end hosts then contact the puzzle server to obtain an ephemeral active IP address for the destination. The puzzle server first asks it to solve a computational puzzle. The source end host solves the computational puzzle and sends it to the puzzle server. The puzzle server returns an ephemeral active IP address of the destination, which can be used to receive service. Our JavaScript mechanism in § V-B enables source end hosts to access any Mirage protected web server without installing any software.

## V. DEPLOYMENT DETAILS

Next, we describe extensions to the basic Mirage design that simplify deployment and enhance security/performance.

### A. Puzzle Server

**Eliminating puzzle verification:** In the Mirage design described so far, the puzzle server issues the computational puzzle seed to the clients, verifies that the clients have solved the puzzle correctly, and sends an active IP address to the clients. We can simplify the complexity at the puzzle servers by having the active IP addresses be the solution to the computational puzzles. This way, the puzzle servers act purely as computational puzzle seed generation and distribution servers. Since IP addresses are puzzle solutions, notice that the destination end-host must be able to efficiently compute the puzzle solutions, since it would need to put the results of the computational puzzle in the network ACLs. We term this feature to be a *trapdoor computational puzzle*. We now provide constructions for such a trapdoor puzzle. As before, let us assume that the destination end host and the puzzle server share a key. The destination end host and the puzzle server can compute the set of active IP addresses as before, i.e.,

$$IP(i) = PREFIX || H(ENC_{KEY}(i || TIME))$$

Let us denote the first 64 bits of the destination end host’s IP address as the *PREFIX*. We denote the last 64 bits of  $IP(i)$  as  $IP_{SUFFIX}(i)$  (equal to  $H(ENC_{KEY}(i || TIME))$ ). Let  $KEY_i$  be a symmetric key. Let  $R$  be a random number. Now, the puzzle server first computes the following:

$$ENC_{KEY_i}(R || IP_{SUFFIX}(i)) = CIPHER_i$$

Next, the puzzle server issues the following to the client a) the *PREFIX*, the  $r$  bit random number  $R$ , the cipher-text of the encryption  $CIPHER_i$ , all but the last  $d$  bits of  $KEY_i$ , where  $d$  is the difficulty level of the puzzle. The job of the client is to iterate through each of the  $2^d$  possible values of  $KEY_i$ , decrypt the cipher-text  $CIPHER_i$ , and check if the first  $r$  bits of the decryption match the random number  $R$ . Upon finding a match, the client can uncover a single IP address by appending the last 64 bits of the decryption to the *PREFIX*. Figure 2 illustrates this computational puzzle design.

**Honest but curious puzzle servers:** We now extend Mirage to use the puzzle servers as untrusted (honest but curious) puzzle distribution servers.<sup>3</sup> The main idea is that the destination end host can communicate the puzzles directly to the puzzle server, so that puzzle server does not have to generate the puzzle. Moreover, by using the previous extension (§ V-A) where the active IP address is the solution to the puzzle, the puzzle server does not need to do any verification, eliminating the need for cryptography at the

<sup>3</sup>Active attacks by puzzle servers, such as returning incorrect JavaScript to source end hosts, can be detected by the victim server (by anonymous querying), and the malicious puzzle servers can be blacklisted.



puzzle server. The destination end host can simply choose a random set of active IP addresses, generate the puzzles as in the previous extension, and store them at the puzzle server. In this fashion, the puzzle server does not need to be trusted with the destination end host’s active IP addresses. Note that a malicious puzzle server could share the puzzles with the attackers ahead of time, enabling precomputation attacks. To mitigate this attack, we propose that the destination end host upload the puzzles at the puzzle server every hopping time interval.

**Eliminating puzzle server as a bottleneck:** So far, we had assumed that the adversary could not launch a DDoS attack on the puzzle server. Since the puzzle servers simply need to serve static puzzles and can be untrusted (due to above mechanisms), the destination end-host can ensure availability of puzzle seeds by simple replication. For instance, the puzzles could be hosted on Akamai or can be widely replicated across cloud service providers such as Amazon S3. The adversary would need to DDoS all of the replicas in order to DDoS the destination. Note that different replicas could either store different puzzles, or use standard CDN approaches to return different puzzle objects to different source end hosts.

### B. End-hosts

**Legacy source end hosts:** We now discuss how source end hosts can take advantage of Mirage protected sites without installing additional software. When the source end host performs a DNS lookup for the destination end host, it is redirected to a DoS-resistant puzzle server (can be hosted by Akamai), which returns JavaScript code.<sup>4</sup> The JavaScript solves the computational puzzle to obtain an active IP address, and issues a cross origin request to that active IP address, either using an *iframe* or a *XMLHttpRequest*. The victim server can set *Access-Control-Allow-Origin* to its own domain, to allow such a cross origin request from the trusted JavaScript, and maintain functionality under the constraints of the browsers same-origin policy (SOP). Sites that disallow framing for defending against clickjacking can set *Frame-Options: Allow-From:* to their own domain. To enable subsequent requests from the client, the JavaScript can set the *base* tag of relative reference links to the active IP address. In this fashion, any standards compliant browser can view a Mirage protected site. Clients that don’t use JavaScript can be directed to a best effort service. The ability to access Mirage protected websites without installing any software makes our design most suitable for web applications.

**Achieving per-computation fairness:** We note that in the protocol described in § IV, honest source end hosts only solve

a single computational puzzle per flow. This may not result in per computation fairness, since the honest node’s computational resources may be under-utilized. We mitigate this problem as follows: when the honest source end hosts obtain an active IP address for the destination end host (after solving a computational puzzle), they continue to spend additional computational resources, which can be used to derive new active destination IP addresses (by querying for and solving another puzzle). This modification in our protocol results in per-computation fair allocation of resources. We now discuss how source end hosts can load balance their traffic over all available active IP addresses. To switch between available active IP addresses, the JavaScript mechanism discussed above could fetch each image or object in the page via a different active IP address. For larger objects, the JavaScript could make multiple requests for particular *byte ranges* of the content (using HTTP range request header), and assemble the returned contents to form the requested page.

**Handling hopping interval transitions:** In Mirage, we set the hopping interval to 5 minutes, as discussed in § VI. A majority of web traffic flows in the Internet are short lived [12], and will finish within a single hopping interval. For scenarios where a short-lived flow starts close to the hopping point, and for long lived flows, observe that the JavaScript is already solving computational puzzles continuously, to achieve per computation fairness. Based on our assumption of loose time synchronization amongst end hosts, the JavaScript can start to solve the puzzles for the next time period and thus receive uninterrupted service. To prevent unnecessary loss in this scenario, the destination continues to receive traffic on the old set of IP addresses (from the previous hopping interval) for a time threshold. Finally, we note that all cross origin requests are made on IP addresses, and thus DNS caching by browsers does not impede JavaScript’s ability to hop IP addresses.

### C. Network

**Filtering traffic:** The destination (victim) end host’s upstream routers need to filter traffic to all non-active IP addresses for the victim. We propose two strategies for filtering traffic, both of which use existing router interfaces. The first strategy is to use an IRSCP [46]-like management server within the destination’s upstream AS. The destination end host could communicate its list of active IP addresses to the management server, and the management server could instruct routers to drop traffic to the remaining non-active IP addresses via iBGP routing updates<sup>5</sup>. Alternatively, the management server could push configuration files containing ACLs for filtered IP addresses to routers. We now discuss how big the ACL should be to support our protocol. Suppose that the local network is under a DDoS attack, and that

<sup>4</sup>A small minority of users who visit the website in a short time window both before and after attack may be affected by browser DNS caching [27]. We tested browser behavior for this scenario using Chrome/Firefox; we found that the first browser request (after attack) is timed out (cached IP address is unresponsive), but the subsequent reloads are successfully redirected to the puzzle server.

<sup>5</sup>Note that iBGP routing updates are local to the AS and do not cause routing instability.

network supports hundreds of thousands of users within the time duration of the hopping interval (set to 5 minutes), typical of large data center environments [15]. Also suppose that we are interested in defending against a botnet with 100 000 bots. In this scenario, it suffices to install an ACL with a few hundred thousand entries (estimated using prior usage history) at the victim’s upstream ISP (assuming that it is the bottleneck). Current routers can already support millions of ACL entries [36], and would not need to be upgraded in this scenario. On the other hand, to defend against larger botnets, or when the deployment happens further upstream, the ACL size could reach tens of millions of entries, in which case existing filtering mechanisms may need to be scaled (which the operator can do by installing more memory at the router, rather than requiring cooperation from the router vendor). We propose an adaptive version of Mirage which can further reduce filter table size in Appendix IX.

**Fair sharing:** Routers already support per-destination fair queueing, and this option can also be manipulated by a configuration files pushed by a management device. For example, the Cisco secure policy manager [13] can read in high level description of policies for a network, and translate them into low level specifications. When a destination end host is under a DDoS attack, it can request its upstream AS to enable per-destination fair queueing, who can then automatically publish the corresponding configurations to routers. Alternatively, if desired, the upstream AS could eliminate the need for a policy manager device by leaving fair queueing always enabled for the customer (some ISPs already run similar QoS mechanisms to improve service for their customers and their own traffic). Observe that the state required to support per destination fair queueing at an upstream router only depends on the number of flows towards downstream destination IP addresses in a short time interval (and not the total downstream IP address space). Finally, we note that our design does not require perfect fair queueing. Thus various traffic monitors located in the upstream AS could check if some active IP addresses are receiving more traffic than others, and if so, then push either a iBGP routing update or an ACL update to block those active IP addresses.

**Supporting IPv4:** Mirage is ideally suited for IPv6, given IPv6’s large address space. While implementing our architecture in IPv4 has some challenges, it is feasible. We consider multiple scenarios for IPv4. In the first scenario, the source end host supports only IPv4, but the destination server and its upstream provider (which is most likely to do filtering) support IPv6. In this case, the source end hosts need to run an IPv4-IPv6 translator, but the network continues to function as before. We note that Teredo IPv6 tunneling [44] is built into Windows and is available on Linux and Mac OS X. In the second scenario, the source end host supports only IPv4 and the destination end host’s network provider also supports only IPv4. In this scenario, the destination end host can set up a

tunnel to another remote IPv6 provider. To prevent attackers from targeting its IPv4 address, the destination end host can request its local provider to filter all access to its IPv4 address (except from its IPv6 tunnel). Now, the victim can advertise its IPv6 address and perform IP address hopping as before. As in the above scenario, source end hosts use an IPv4-IPv6 translator, and the network does not need any software or hardware updates. Alternatively, the victim server could purchase service directly from an IPv6-enabled provider.

## VI. SECURITY ANALYSIS

Next, we analytically evaluate the security of our design.

**Network scanning attacks:** A possible threat to Mirage is the use of network scanning attacks to uncover the set of active IP addresses in use by an end host. Due to the large range of possible IP addresses for an end host (64 bits), such attacks are already quite costly. The set of active IP addresses changes after every hopping time interval (which we set to 5 minutes); thus the attacker has to scan the entire range within the hopping time interval. Consider two attack scenarios: (a) the size of botnet owned by the attacker is 20,000 (the average size of modern botnets [39]) and (b) the size of the botnet is 1 million nodes (an extreme scenario). Now, if the bots have 1Mbps links to their provider, each bot would be able to send only about  $2^{25}$  ping packets over the 5 minute interval. Thus, in the two attack scenarios, the attacker would only be able to scan an insignificant fraction of the range of IP addresses for an end host -  $2^{-28}$  and  $2^{-23}$  respectively.

**Brute-force DoS attacks:** The attacker could simply send attack traffic to the victim using a random interface (subnet) address for each packet, with the idea that a fraction of its traffic would be able to bypass router filters and congest the victim link. With perfect filtering of non-active IP addresses at the routers, such brute-force DoS attacks would be completely ineffective as the probability of a random IP address being an active IP address is very small. E.g., suppose there are about 5000 client nodes per 5 minute interval per destination end host. Then, the set of active IP addresses needed to contain about 25000 and 1,005,000 entries respectively (for the two attack scenarios discussed above), resulting in the success rate of a random attack of only  $1.3 \cdot 10^{-15}$  and  $5.4 \cdot 10^{-14}$  respectively.

**Theoretical Results:** Next, we present key analytical results which illustrate that (a) Mirage’s computational puzzle mechanism is secure, (b) Mirage provides per computation fairness, and (c) Mirage is able to gracefully deal with compromised routers.

*Hypothesis 1:* Puzzle scheme security: For a computational puzzle with difficulty level  $d$ , the attacker has negligible advantage in obtaining active IP addresses compared to honest sources.

*Proof:* We can model the AES block cipher as a pseudorandom permutation. If the attacker has any advantage



over an honest source end host's brute force strategy for solving the computational puzzle, then the attacker would be able to break the pseudorandom permutation as well. Thus, the best strategy for the attackers is a brute force through the  $2^d$  possible keys, which is identical to the honest client strategy.

*Hypothesis 2:* Per computation fairness: Let  $C_A$  and  $C_H$  denote the computational resources for the attacker and honest sources respectively. The honest source end host's share of the victim bandwidth is  $\frac{C_H}{C_H+C_A}$ .

*Proof:* From Theorem 1, we know that the attacker's advantage over honest sources in solving the computational puzzles is negligible. We have already shown that network scanning attacks to uncover active IP addresses are ineffective. Also, from Section V-B, source end hosts fully utilize their computational resources by continuing to solve computational puzzles even after obtaining a single active IP address for the destination. Thus after any amount of time  $t$ , the number of active IP addresses known by the attackers and the honest clients are proportional to  $C_A$  and  $C_H$  respectively. Finally, in our protocol, the network performs per destination fair queuing, resulting in fair sharing of resources amongst active IP addresses. Thus the honest source end hosts get  $\frac{C_H}{C_H+C_A}$  of the total bandwidth.

*Hypothesis 3:* Impact of compromised routers: Let a set of compromised routers carry a fraction  $f$  of the legitimate traffic towards a destination end host under attack. If the set of compromised routers collude with the adversary, then the honest sources' share of the victim bandwidth is  $\frac{C_H \cdot (1-f)}{C_H+C_A}$  of the total bandwidth.

*Proof:* In Mirage, malicious routers that handle a fraction  $f$  of the legitimate traffic towards the destination end host can snoop on the legitimate traffic towards the destination end host and uncover its active IP addresses. Moreover, in addition to snooping on the legitimate traffic, the malicious routers can drop the legitimate client traffic towards the destination to increase the attacker's bandwidth share. Thus the attackers will be able to obtain a fraction  $\frac{C_A+f \cdot C_H}{C_A+C_H}$  of the total bandwidth, while the honest source end hosts obtain a fraction  $\frac{C_H \cdot (1-f)}{C_H+C_A}$  of the total bandwidth.<sup>6</sup>

## VII. EVALUATION

In this section, we quantify the attack resilience of Mirage using a prototype implementation and an ns-2 simulation, compare Mirage with other DoS defenses, and perform Internet bandwidth measurements to determine the extent to which Mirage must be deployed in order to provide protection against various levels of DoS.

<sup>6</sup>We note that compromised routers in the destination's upstream AS also have the potential to reveal the access control list to the adversary, but the upstream AS can easily detect and isolate such compromised routers by inserting a few spurious entries in the ACL and checking for traffic towards those destination IP addresses.

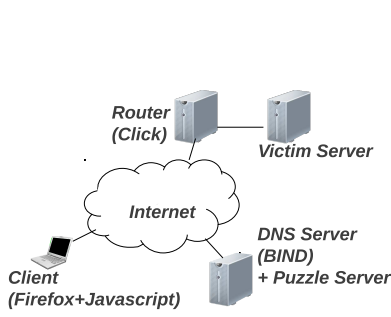
### A. Prototype Implementation

To evaluate Mirage's performance and deployability, we built a prototype implementation of Mirage. We did not attempt to optimize the prototype system; rather, our goal is to verify the design and evaluate its deployability. To evaluate our prototype, we constructed a *Mirage-enabled web service*, which we call the *victim server*, and configured a set of clients to access that web service.

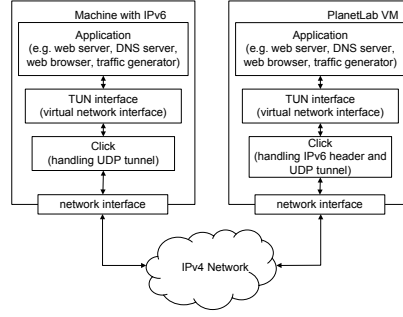
**Client side:** We implemented the JavaScript program described in § V-B to enable the client to "hop" addresses without requiring changes to the client's browser or operating system. This JavaScript is located at the puzzle server as shown in Figure 3(a). When the DNS directs a client to the puzzle server, the client fetches and solves the puzzle by running the JavaScript provided by the puzzle server. These operations are transparent to the user; the user simply directs its browser to the victim's domain name.

**Server side:** The server-side components, comprised of the victim server, router, DNS server, and puzzle server, provide *transparent* protection for a victim server. By transparent, we mean that the user of a client machine need not install any additional software. We used BIND [11] to implement the DNS server. We defined our own zone (miragev6.org) and created a probing tool that regularly sends pings to the victim server to detect when the victim server is under attack. Initially, www.mirage v6.org resolves directly to the victim's IP address. Once the probing tool fails to receive a certain number of echo responses, it changes the DNS record so that www.miragev6.org resolves to the puzzle server's IP address (which, in our prototype, is the DNS server). We set the TTL for the DNS record to a small value so that an entry cached before the victim was under attack will be quickly corrected after the attack. When the victim is under DoS attack and the client tries to access www.miragev6.org, the client receives the IP address of the puzzle server. The client connects to the puzzle server and fetches JavaScript code which includes a puzzle. Our implementation uses the scheme in § V-A, using AES as the encryption algorithm.

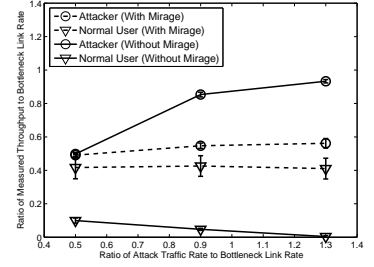
**Experimental setup:** To investigate Mirage's operation in the wide-area, we conducted experiments in the Internet using the PlanetLab testbed. Figure 3(b) shows how we set up the network. We selected random (lightly CPU-loaded) nodes to run our experiments. We selected one PlanetLab node to act as the client, and one PlanetLab node to run the server, router, puzzler server, and DNS software. Since PlanetLab hosts do not directly provide IPv6 compatibility, for our IPv6 experiments we used an IPv4 encapsulated tunnel. We found that standard tunnel (e.g. GRE) packets are blocked by firewalls of some PlanetLab networks, so we used UDP tunnels for our experiments. We implemented this tunnel with the TUN [45] virtual interface and Click [14]. In an IPv6-enabled machine, outgoing IPv6 packets are delivered to TUN device which lets Click capture the packets. Click



(a) Prototype implementation architecture.



(b) Experimental setup.



(c) Mirage guarantees normal users' throughput even with high-rate attackers.

then encapsulates the packets with UDP tunnel. Because PlanetLab nodes lack kernel support for IPv6, we assigned IPv4 addresses to the TUN device and used Click to translate between IPv4 and IPv6 addresses.

### B. Performance Evaluation

In this section, we evaluate how well Mirage can defend against possible attacks. We do not argue that we consider all possible attacks; rather, our intention is to quantify Mirage's attack resilience. We consider two attacks: one that exhausts network bandwidth and one that exhausts hopping IP addresses.

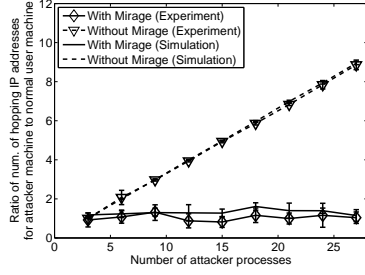
**Bandwidth exhaustion attack:** By increasing its sending rate, or the number of flows it sends, an attacker can increase its traffic rate to overwhelm normal users' traffic. We examined the effect of the attack traffic rate on benign traffic by setting up ten benign TCP traffic flows and one UDP attack traffic flow that share a single bottleneck link to the victim. We measured the throughput of the benign and the attack traffic, and compared them with and without Mirage. Figure 3(c) shows the results when we vary the attack traffic rate between 0.5 to 1.3 times the bandwidth of the bottleneck link. Without Mirage, as the attack traffic rate increases, the normal user's traffic rate decreases; with Mirage, fair queueing drops the attackers' excess packets rather than the normal user's traffic.

**Hopping IP address exhaustion attack:** Since Mirage uses fair queueing per hopping IP address, an attacker can request a large number of hopping IP addresses to prevent normal users from getting hopping IP addresses, thus preventing normal users from transmitting to the victim. In order to evaluate Mirage's performance under a large number of attackers, we use the ns-2 simulator [37]. To parametrize the simulation, we developed a computational model and used Internet experiments using a small number of attacker processes to estimate the parameters for our model.

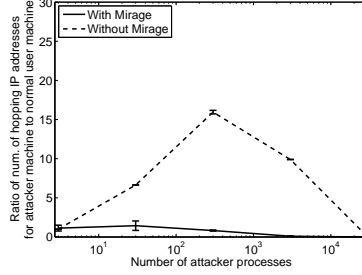
Our Internet experiments used three machines to behave as an attacker and one machine to behave as the normal user. We implemented a client using C code to access the Mirage-enabled web service. To demonstrate the performance of our JavaScript extension, we also considered Firefox to be the client. Both normal users and attackers solve puzzles

as quickly as possible in order to obtain hopping addresses; however, in normal user machines, a client process generates IP addresses, whereas on attacker machines, we vary the number of client processes that are generating IP addresses. In our prototype implementation, the puzzle server uses hopping IP addresses released from a leaky bucket and assigns it to the puzzle solution with the highest difficulty. A process that is not chosen will increase its puzzle difficulty and try again. Without Mirage, the puzzle server grants all requests for IP addresses. We measured the number of hopping IP addresses obtained by the attacker machines and the normal user machine. Figure 4a shows the ratio of the number of hopping IP addresses per attacker machine to the number of hopping IP addresses of normal user machine when we use Firefox as a client. Without Mirage, as the number of attacker processes increases, the attacker machines can get a larger number of hopping IP addresses, which corresponds to a large number of victim machine accesses. Note that for Firefox processes, the growth rate becomes smaller as the number of processes increases because the additional processes will saturate the machine's computational power. We can see that with C code, the saturation of computation power is reduced. With Mirage, this computational limit is reached much more quickly, because a node's ability to get IP addresses is computationally limited.

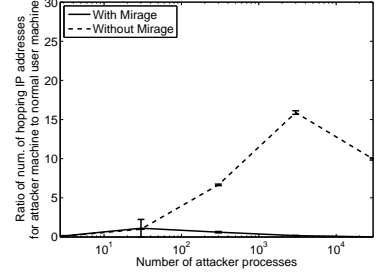
We used this experimental data to establish a model in ns-2. We used a dumbbell topology with RTT equal to that from our experimental study. When a node solves a puzzle solution, it takes an amount of time to generate a solution given by  $\frac{B+N \times E}{C}$  where  $B$  is the cost (in cycles) of running a client process without puzzle calculations,  $N$  represents the number of encryption attempts needed to solve a puzzle, which we model as a normal distribution,  $E$  is the number of cycles taken by each encryption attempt, and  $C$  is the speed of the CPU. Without Mirage, a node takes time  $B/C$  to resolve the victim's IP address. We varied the values of  $B$ ,  $N$ , and  $E$  to find parameters that best matched our experimental study; that is, ones that minimized the variance between experimental and simulation results. Figure 4a shows how well the two sets of data match. With these simulation parameters, we performed simulation studies to show how the number of attacker processes affects the



(a) Implementation study with a small number of attacker processes (3 attackers)



(b) Simulation study with a large number of attacker processes (3 attackers)



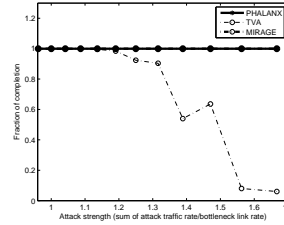
(c) Simulation study with a large number of attacker processes (30 attackers)

Fig. 4. Defense against hopping IP address exhaustion attack.

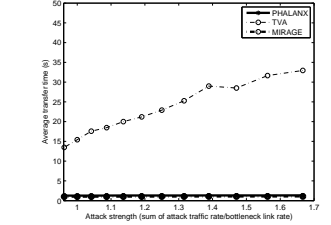
number of hopping IP addresses that the attacker machine obtained. Figures 4b and 4c show the results when the number of attacking machines is 3 and 30, respectively. We vary the number of attacker processes from 3 to 30000. In all cases, we can see that with Mirage, an attacker cannot obtain as many hopping IP addresses as in the case without Mirage. Note that the attacker performance dips beyond a threshold number of attacker processes due to resource constraints at the attacker machines.

### C. Comparative Study

We compare the effectiveness of different DoS defense mechanisms: Phalanx [17], TVA [53] and Mirage, using the ns-2 simulator. For Phalanx, we developed ns-2 code and for TVA, we used the code developed by authors of TVA [53]. For Mirage, we used DRR as our fair queueing mechanism. We followed the methodology in the work of Liu et al. [32]. We used the IPv4 Routed /24 Topology Dataset [26] to derive the AS-level topology, which has links annotated with round trip times at each hop. We scaled down the AS-level topology by 1/200 since the memory of our simulation machine does not support the full Internet-scale AS-topology. In our simulated environment, ten legitimate TCP senders try to send 50 files each, of size 2KB. Each sender immediately starts sending a file after its previous file transfer is done. We measured the fraction of successful file transfers and the average time to deliver files. While legitimate senders send TCP traffic, attackers send UDP traffic. All legitimate and attacker traffic goes through the same bottleneck link to a destination. For the first simulation, we intended 10K attackers in the Internet-scale. Hence in our simulation scaled down, the number of simulated attackers is 50, and each attacker sends 10Kbps of UDP traffic. Figure 5 shows results when we vary the bottleneck link bandwidth from 300Kbps to 520Kbps. For the second simulation, we vary the number of attackers from 5 to 500 fixing the bottleneck link bandwidth as 500Kbps. Figure 6 shows the results in this case. Overall, we find that Mirage achieves its deployment benefits while providing comparable results to Phalanx and TVA across a variety of attack strengths.

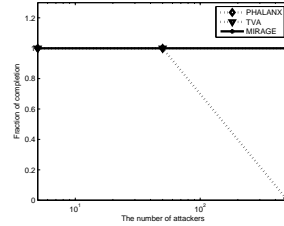


(a) File transfer completion

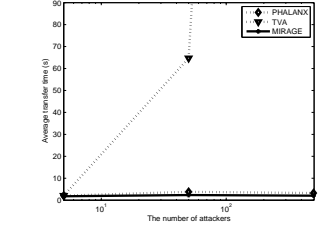


(b) Average transfer time

Fig. 5. Comparative study varying link bandwidth



(a) File transfer completion



(b) Average transfer time

Fig. 6. Comparative study varying number of attackers

### D. Bottleneck Measurements on PlanetLab

The goal of this study is to compute the *pushback links*: the nearest links to the victim server that are sufficient to block the attacker's traffic while still letting non-attack traffic through. Computing these links sheds light on how close to the network edge (router hops/AS hops) Mirage can be deployed while still realizing its benefits. To gain information about link bandwidths in today's Internet, we ran *pchar*, an open-source reimplementation of pathchar, from 336 randomly-selected PlanetLab nodes to the top 100 websites from alexa.com. We constructed a target IP address set by selecting a single IP address per nslookup query (the first IP address). We ran *pchar* using 32 repetitions per hop, and testing each hop with 300 byte increments. We ignored route changes during the course of a *pchar* experiment, by setting the *-c* flag. To improve accuracy of *pchar* data, we also ran the *pathneck* [25] bandwidth probing tool over the same set of links. Unlike *pchar*, *pathneck* measures the available bandwidth rather than the capacity of links. Any capacity estimate lower than the available bandwidth estimate was considered an error. In addition, we combined multiple experiments for improved accuracy. For the links closer to



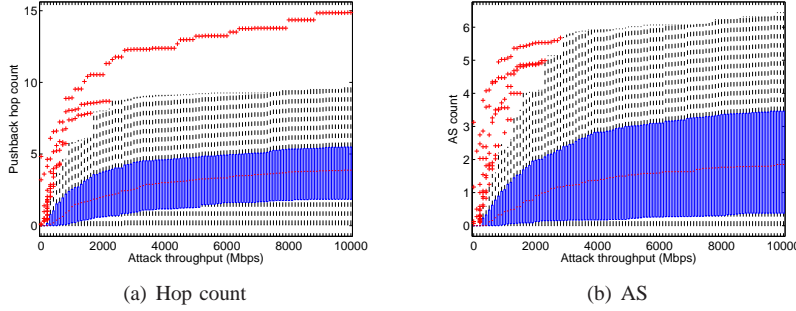


Fig. 7. Running Mirage at only the server's local network and ISP provider (first AS-level hop) is sufficient to achieve most of Mirage's benefits.

the target hop, we may have several pchar estimates from different PlanetLab nodes. In this case, we can improve accuracy by considering the median value of the estimates to be the final estimate of the link capacity (and thus mitigate the effect of outliers). If all pchar estimates for a link were erroneous, then for our analysis, we assume that such hops are not the bottlenecks for the DDoS attack.

**Data analysis methodology:** For our analysis, we assume that the distribution of malicious nodes in the network (bot nodes) is similar to the distribution of PlanetLab nodes selected for our experiments. While our assumption is only a heuristic, our methodology is one way to shed light on the problem (without actually carrying out a DDoS attack). For each target IP, we first combine the results of all PlanetLab experiments, and merge them into a single network map, rooted at the target IP. We annotate links in the map with the fraction of paths traversing that link, and the median of the capacity estimates. Since we assume that the distribution of PlanetLab nodes is similar to the distribution of malicious nodes, we first approximate the amount of attack traffic traversing each link, for particular attack throughput values. For example, if the total attack throughput is 1Gbps, and 10% of the PlanetLab paths (probes from PlanetLab nodes to the victim server) traverse that link, then the amount of attack traffic traversing that link is  $0.1 * 1\text{Gbps} = 100\text{Mbps}$ . Using the bandwidth annotated network map, it is now possible to compute the set of optimal pushback links. Finally we compute the weighted mean of the router level hop counts for the set of pushback links, where the weight indicates the fraction of attack traffic traversing that router. In addition to computing router-level hops for the bottleneck points, we also consider AS-level hops. For this purpose, we compute the IP to AS mapping using a BGP RIB snapshot collected from Routeviews [40]. Our study does not account for the scenarios where an AS has multiple AS numbers or when prefixes within a single AS number have multiple owners.

**Experimental Results:** Figure 7 depicts the distance to pushback links as a function of the attack traffic using a boxplot, both in terms of the router and AS hop counts. When the attack traffic is less than 2 Gbps (vast majority of real world attacks in 2011 [30]), the median distance to the pushback links is less than 2 router hops and 1 AS hop.

Thus running Mirage at only the victim's local network and ISP provider is sufficient to achieve Mirage's benefits in this scenario. Even when the attack throughput is 10 Gbps, the median distance to the pushback links is only 4 router hops and 2 AS hops.

## VIII. DEPLOYMENT

Like previous DDoS mitigation schemes, our design requires changes to certain components within the network. However, our work differs from "clean slate" approaches which require large-scale changes to the Internet infrastructure to achieve their benefits. In particular, our experimental results show that only the victim server's local network and its upstream ISP can deploy Mirage to defend against moderate scale attacks. Moreover, Mirage is incentive compatible. Since the victim has a business relationship with its upstream ISP, this may spur economic benefits for adoption [19]. Incrementally deploying Mirage at ISPs that are further upstream increases Mirage's resilience against attack. Mirage requires the following modifications to existing systems:

**ADNS:** Mirage requires changes to the server's authoritative DNS server, to redirect clients to the puzzle server, which can be hosted by *existing* services such as Akamai or Amazon S3. Since the authoritative DNS server is typically owned and operated by the service provider, many deployed systems (e.g., Akamai) leverage the ADNS as an easy-to-modify location to instrument their designs.

**Server:** Mirage can be deployed as a bump in the wire solution at the server; existing software does not need to be modified. For example, Mirage software can simply bind the server's hopping IP addresses to 0.0.0.0. Alternately, Mirage can also be deployed as a reverse proxy.

**Server's upstream network(s):** Mirage also requires changes to networks. However, Mirage does not require widespread adoption in order for servers to benefit. Changes to network infrastructure that is under direct influence of the service provider, such as its own network, and ISPs that it directly pays for service, suffice to defend against moderate scale attacks. In addition, these network changes do not rely on router vendors to incorporate new changes into their software, and can instead be realized through configuration changes. However, depending on the point of deployment,

<i>Mechanism</i>	<i>Mirage</i>	<i>NetFence [33]</i>	<i>TVA [53]</i>	<i>Portcullis [38]</i>	<i>Phalanx [17]</i>
<i>Filtering</i>	<b>IP dest based</b>	Capability based	Capability based	Capability based	Capability based
<i>Router communication</i>	<b>Out of band</b>	New header	New header	New header	New header
<i>Fair queuing/rate limiting</i>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>
<i>Puzzle distribution</i>	Yes	<b>No</b>	<b>No</b> (attack [38])	Yes	Yes
<i>Bandwidth-based fairness</i>	<b>No</b>	<b>No</b>	<b>No</b>	<b>No</b>	Yes
<i>Router Cryptography</i>	<b>No</b>	Yes	Yes	Yes	<b>No</b>
<i>Other</i>	<b>None</b>	Passport	Path identifiers	<b>None</b>	Traffic forwarding via overlay network

TABLE I  
IN-NETWORK SUPPORT REQUIREMENTS FOR DDoS DEFENSE SCHEMES.

<i>Deployment details</i>	<i>Mirage</i>	<i>NetFence [33]</i>	<i>TVA [53]</i>	<i>Portcullis [38]</i>	<i>Phalanx [17]</i>
<i>Source upgrades</i>	<b>Optional</b>	Required	Required	Required	<b>Optional</b>
<i>Destination upgrades</i>	<b>Required</b>	<b>Required</b>	<b>Required</b>	<b>Required</b>	<b>Required</b>
<i>Bottleneck router(s) upgrades</i>	<b>Required</b>	<b>Required</b>	<b>Required</b>	<b>Required</b>	<b>Required</b>
<i>Other router(s) upgrades</i>	<b>None</b>	Yes	Yes	<b>None</b>	Yes (Overlay member)
<i>Puzzle distribution</i>	Akamai/DNS	<b>None</b>	None (attack)	Akamai/DNS	Akamai/DNS
<i>Router upgrade type</i>	<b>Configuration changes (may need more memory)</b>	Software and hardware changes (by vendors)	Software and hardware changes (by vendors)	Software and hardware changes (by vendors)	Software and hardware changes (by vendors)
<i>Other</i>	<b>None</b>	<b>None</b>	<b>None</b>	Trusted authority	Overlay network

TABLE II  
AFFECTED DEPLOYMENT LOCATIONS FOR DDoS DEFENSE SCHEMES.

the ACL mechanism may need to be scaled (which can be done by the operator by installing more memory, rather than needing to convince the vendor to change the router’s hardware/software design. These configuration changes can be automated through the use of an IRSCP [46] to install ACLs within the network. Mirage can alternatively leverage OpenFlow, with Mirage’s functionality implemented within the NOX controller. Several large ISPs have already begun offering commercial services to allow customers to install prefixes for blacklist filtering [16].

#### A. Comparison with other approaches

Next, we contrast the deployment challenges of Mirage with previous proposals, in terms of what changes need to be made to the infrastructure (network primitives), and which players in the network need to instrument those changes (administrative boundaries).

**Adding new primitives to the network:** Deploying new functionality in the network becomes easier when the changes to existing devices are small. Protocols that require new primitives in routers require coordination with (and across) network vendors to be realized. Table I shows the type of changes required to network devices to deploy several recently-proposed DDoS mitigation schemes. First, NetFence, TVA, Portcullis, and Phalanx perform filtering based on some form of a capability. This requires modifications to the software of Internet routers to process and modify this capability. In contrast, Mirage performs filtering based on the destination IP address, and communicates ACL entries to routers using legacy protocols (iBGP feeds or configuration changes via sessions to an IRSCP). Next, Mirage, Portcullis, and Phalanx require computational puzzles (proof of work) in order to defend against attacks targeting the request channels. TVA does not use computational puzzles, but this makes it

vulnerable to the Denial of Capability (DoC) attack [38]. NetFence requires defenses to prevent the IP spoofing attack (such as Passport) to be widely deployed in order to secure the request channel. In addition to computation-based fairness using computational puzzles, Phalanx also implicitly adopts bandwidth-based fairness [48], an approach which is known to induce congestion collapse in the network. Finally, the non-Mirage mechanisms require additional functionality from the network, such as 1) cryptography at the routers for NetFence, TVA, and Portcullis, 2) insertion of path identifiers for TVA, and 3) overlay network to forward traffic for Phalanx.

**Deploying changes across administrative boundaries:** Deploying new functionality in the network becomes easier when fewer participants have to cooperate in deploying those changes. Protocols that require clients to install new software, or require widespread upgrades to routers can complicate deployment. Table II describes the location where changes would need to be realized within the network. First, only Mirage and Phalanx do not require source hosts to upgrade. Second, all mechanisms require destinations to upgrade, and also require deployment at the bottleneck routers (points of congestion). However, TVA and NetFence also require other routers to upgrade, for inserting path identifiers and for defending against IP spoofing; this means that it is not possible for an ISP to unilaterally deploy NetFence. Similarly, Phalanx also requires either an external overlay network to absorb the DDoS attack, or requires additional routers to upgrade and participate in the overlay scheme. Portcullis requires all Internet routers to trust a single entity. Finally, Mirage, Portcullis, and Phalanx require an additional deployment component to distribute puzzles. However, we note that puzzle distribution in Mirage, Portcullis and Phalanx does not require global cooperation from ISPs. A Victim

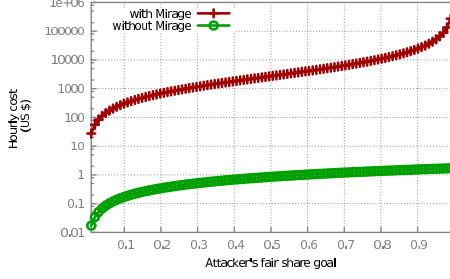


Fig. 8. Hourly DDoS attack cost as a function of desired share of victim’s capacity. Mirage raises the attack cost by several orders of magnitude.

end host can easily use existing CDNs like Akamai, which can be given financial incentives to participate. Finally, we note that Mirage is the *only* scheme in which functionality required from the routers is already deployed in today’s routers, though this functionality (ACLs) may need to be scaled depending on the point of deployment. Routers already have support for ACLs and fair queuing, which can be enabled with relatively straightforward configuration commands. Scaling memory requirements for ACLs can be done by the operator by installing more memory at the router, rather than requiring router vendor’s cooperation.

## IX. DISCUSSION

Here, we discuss some ramifications of our design.

### A. Applicability to non-web applications

Mirage is ideally suited to defend against web applications, since its JavaScript mechanism obviates the need for client software. However, Mirage can also be used to defend against non-web applications, at the cost of upgrading clients. The JavaScript mechanism can be replaced by client software that provides equivalent functionality of fetching/solving puzzles. We note that even in this setting, Mirage provides stronger deployment properties compared with previous work, since it provides security benefits when only the victim’s upstream ISP cooperates (as illustrated in Table II).

### B. Cost analysis using EC2 pricing

Next, we perform a cost analysis of launching a DDoS attack by considering (a) Amazon EC2 prices for computation and data transfer [1], and (b) a data center environment which serves hundreds of thousands of users in short time periods [15] as the victim server. Let us suppose that the victim has a 1 Gbps link to its service provider, and that the computational capability of each honest user is 1 EC2 compute unit (equivalent to 1.0-1.2 GHz 2007 Xeon processor). Without Mirage, the cost of obtaining a fraction  $x$  of the victim’s network capacity is proportional only to the required EC2 data transfer. With Mirage, the attacker is forced to spend extra resources for solving computational puzzles. Figure 8 depicts the hourly cost of launching a DDoS attack in US dollars, as a function of the attacker’s desired share of the victim’s network capacity. We can see that

Mirage raises the attack cost by several orders of magnitude. In future work, we will extend this analysis by considering botnet rent prices for computation and bandwidth.

### C. Adaptive network defense

To improve performance under small and moderate sized attacks, we describe an adaptive mechanism that utilizes network support in proportion to the strength of the attack. In particular, we organize the active set of IP addresses as a binary tree, with nodes in the tree representing IP addresses, and the number of leaves equaling the number of potential clients. Levels in the tree represent priorities: IP addresses at level  $i$  are assigned half the priority as compared to IP addresses at level  $i+1$ . We modify our puzzle server to return multiple IP addresses, one from each level in the tree. Source end hosts first try the IP address corresponding to the root of the tree (level 0), and if they are unable to get good service, they adaptively switch to the next level in the binary tree. This has the advantage that the router only needs to maintain *per aggregate state* in the access control lists while performing fair queuing. Similarly, the ACL entries could also be added adaptively by the destination end host - it starts off by adding the root of the tree, and then progressively adding next level IP addresses as needed. This reduces the ACL list size.

### D. Authorized users and classful services

We can extend our architecture to enable destination end hosts to provide higher priority to authorized users. Firstly, victim servers could communicate private IP addresses to authorized users using out-of-band communications. Secondly, authorized users could use Mirage to authenticate to the victim server and receive private IP addresses, or *hints* to solve computational puzzles. Finally, authorized users could authenticate themselves to the puzzle servers, using a server-supplied cookie (possibly cached from last visit by the browser). The authentication could be performed in zero knowledge, with no information about client credentials being revealed to the puzzle server. Upon authentication, the puzzle server could provide either private IP addresses or hints for puzzle to the authorized users.

We note that our architecture can enable a useful scenario where a sensitive server’s IP address is visible only to authorized users. Here, the server would need to supply the cookie to authorized users via an out of band channel, and the puzzle server would respond only after authentication. The server’s IP address is practically invisible to the rest of the Internet, and is thus largely immune to threats like bot infections due to random IP scanning.

### E. Insider collusion attacks

A potential concern in Mirage is the collusion between a compromised end host in the victim network and the attacker. Suppose that the victim network has two nodes. The compromised end host could install an ACL at the upstream



AS with  $k$  entries, with the attackers having full knowledge of the active IP addresses for the compromised host. The attackers could then send traffic to the  $k$  IP addresses listed in the ACL. Due to per destination fair queueing at the upstream routers, the attackers would gain an overwhelming share of the traffic at the victim link and DoS the honest end host. In practice, the operator can deal with this through appropriate network management. One management policy to solve the problem could be to bound the size of the ACL per victim.

#### F. Prefix/path/location hopping

While our design focuses on hopping of IP addresses, it may be possible to apply this technique to other components of network protocols. For example, it may be possible to perform *prefix hopping*, by having router prefixes changing with time, *path hopping*, by changing the set of Internet paths used to route to a destination, and *location hopping*, by leveraging virtual machine migration techniques to relocate services on-demand in the presence of a DoS attack. These techniques may assist in providing improved security properties in present and future Internet architectures.

### X. CONCLUSION

Mirage uses an analog of *frequency hopping* from wireless networks: hosts vary their IP address through a pseudorandom sequence to evade attacks by unauthorized hosts. From theoretical analysis, simulations, and experiments from a prototype implementation, we find that Mirage achieves comparable performance to previous DDoS mitigation mechanisms. However, Mirage has improved deployment properties over previous designs. Mirage does not require an external overlay network, or a trusted Internet authority, or defenses against IP spoofing. In particular, Mirage is incrementally deployable; a victim server and its upstream ISP can deploy Mirage and defend against moderate scale attacks, without requiring source end hosts to install any software.

### ACKNOWLEDGMENTS

We are grateful to Prateek Saxena and Devdatta Akhawe for discussions about our JavaScript mechanism. Robin Sommer, Vern Paxson, Brian Tierney, and Chin Guok provided us with helpful feedback.

### REFERENCES

- [1] Amazon EC2 pricing. <http://aws.amazon.com/ec2/pricing/>.
- [2] Internet assigned numbers authority. <http://www.iana.org/>.
- [3] World IPv6 day. <http://isoc.org/wp/worldip6day/>.
- [4] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Trans. Internet Technol.*, 5, 2005.
- [5] M. Abu-Rgheff. *Introduction to CDMA Wireless Communications*. Academic Press, 2007.
- [6] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: using hard AI problems for security. In *EUROCRYPT*, 2003.
- [7] Akamai. [www.akamai.com](http://www.akamai.com).
- [8] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet protocol (AIP). *SIGCOMM Comput. Commun. Rev.*, 38(4), 2008.
- [9] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. *Comput. Netw.*, 51(12), 2007.
- [10] K. Argyraki and D. R. Cheriton. Active Internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX ATC*, 2005.
- [11] BIND— Internet systems consortium. <http://www.isc.org/software/bind>.
- [12] N. Brownlee and K. claffy. Understanding Internet traffic streams: Dragonflies and Tortoises. *IEEE Communications Magazine*, 2002.
- [13] Cisco secure policy manager. [http://www.cisco.com/en/US/products/sw/secursw/ps2133/prod\\_technical\\_reference09186a00800a9ebc.html](http://www.cisco.com/en/US/products/sw/secursw/ps2133/prod_technical_reference09186a00800a9ebc.html).
- [14] Click. <http://www.read.cs.ucla.edu/click>.
- [15] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *SIGCOMM*, 2011.
- [16] DDoS Defense — Enterprise Business — AT&T. <http://www.business.att.com/enterprise/Service/network-security/threat-vulnerability-management/ddos-protection/>.
- [17] C. Dixon, T. Anderson, and A. Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. In *NSDI*, 2008.
- [18] M. Donner. Cyberassault on Estonia. *IEEE S & P*, 5:4–5, 2007.
- [19] P. Gill, M. Schapira, and S. Goldberg. Let the market drive deployment: a strategy for transitioning to BGP security. In *SIGCOMM*, 2011.
- [20] V. D. Gligor. Guaranteeing access in spite of distributed service-flooding attacks. In *Security Protocols Workshop*, 2003.
- [21] J. A. Halderman and B. Waters. Harvesting verifiable challenges from oblivious online sources. In *CCS*, 2007.
- [22] R. Hinden and S. Deering. Rfc 3513: Internet protocol version 6 (IPv6) addressing architecture. <http://www.ietf.org/rfc/rfc3513.txt>, 2003.
- [23] R. Hinden and S. Deering. Rfc 4291: Internet protocol version 6 (IPv6) addressing architecture. <http://tools.ietf.org/html/rfc4291>, 2006.
- [24] C.-Y. Hong, C.-C. Lin, and M. Caesar. Clockscalpel: Understanding root causes of Internet clock synchronization inaccuracy. *PAM*, 2011.
- [25] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. *SIGCOMM*, 2004.
- [26] The IPv4 routed /24 topology dataset. [http://www.caida.org/data/active/ipv4\\_routed\\_24\\_topology\\_dataset.xml](http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml).
- [27] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting browsers from DNS rebinding attacks. In *CCS*, 2007.
- [28] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, editors. *Moving Target Defense - Creating Asymmetric Uncertainty for Cyber Threats*, volume 54 of *Advances in Information Security*. 2011.
- [29] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. In *NSDI*, 2005.
- [30] Kaspersky-Labs. 57% hike in the power of DDoS attacks. [http://www.kaspersky.com/about/news/virus/2012/Number\\_of\\_the\\_week\\_57\\_hike\\_in\\_the\\_power\\_of\\_DDoS\\_attacks](http://www.kaspersky.com/about/news/virus/2012/Number_of_the_week_57_hike_in_the_power_of_DDoS_attacks).
- [31] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *SIGCOMM*, 2002.
- [32] X. Liu, X. Yang, and Y. Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *SIGCOMM*, 2008.
- [33] X. Liu, X. Yang, and Y. Xia. NetFence: Preventing Internet denial of service from inside out. In *SIGCOMM*, 2010.
- [34] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.
- [35] W. G. Morein, A. Stavrou, D. L. Cook, A. D. Keromytis, V. Misra, and D. Rubenstein. Using graphic turing tests to counter automated DDoS attacks against web servers. In *CCS*, 2003.
- [36] Nanog. <http://www.nanog.org/meetings/nanog39/presentations/bof-report.pdf>.
- [37] NS-2: Network simulator. <http://www.isi.edu/nsnam/ns>.
- [38] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y.-C. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. In *SIGCOMM*, 2007.
- [39] L. D. Paulson. News briefs. *Computer*, 39(4):17–19, 2006.
- [40] University of Oregon Route Views project. <http://www.routeviews.org/>.
- [41] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *CCS*, 2004.
- [42] C. A. Shue, A. J. Kalafut, M. Allman, and C. R. Taylor. On building inexpensive network capabilities. *SIGCOMM Comput. Commun. Rev.*, 42(2).

- [43] A. Stavrou and A. D. Keromytis. Countering dos attacks with stateless multipath overlays. In *CCS*, 2005.
- [44] Teredo: Tunneling IPv6 over UDP through network address translations. <http://tools.ietf.org/rfc/rfc4380.txt>.
- [45] Universal TUN/TAP driver. <http://vtun.sourceforge.net/tun>.
- [46] J. Van der Merwe, A. Cepleanu, K. D'Souza, B. Freeman, A. Greenberg, D. Knight, R. McMillan, D. Moloney, J. Mulligan, H. Nguyen, M. Nguyen, A. Ramarajan, S. Saad, M. Satterlee, T. Spencer, D. Toll, and S. Zelingher. Dynamic connectivity management with an intelligent route service control point. In *INM*, 2006.
- [47] Visa, Mastercard targeted in apparent cyber-attack. <http://news.blogs.cnn.com/2010/12/08/visa-website-down-after-threat-from-wikileaks-supporters/>.
- [48] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS defense by offense. *ACM Trans. Comput. Syst.*, 28(1), 2010.
- [49] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *IEEE S&P*, 2003.
- [50] X. Wang and M. K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *CCS*, 2004.
- [51] Wikileaks under denial of service attack. <http://www.securityweek.com/wikileaks-under-denial-service-attack-ddos>.
- [52] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. *IEEE S & P*, 2004.
- [53] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *SIGCOMM*, 2005.